Introduction to the MIRI Data Models

Steven Beard
UK Astronomy Technology Centre

2nd April 2019

Purpose of the MIRI Data Models

- **JWST data is described by a collection of data models developed by STScI.**
  - Each data model consists of:
    - A Python class (derived from a DataModel base class).
    - A description of the data structure in a structured text file: a YAML schema.
- **The MIRI data models were created to ensure that MIRI data are stored in a format compatible with the JWST data models, as described here:**
  - https://jwst-pipeline.readthedocs.io/en/latest/jwst/datamodels/index.html
- **Most MIRI data models are based on a JWST data model. For example MiriFlatfieldModel is based on FlatModel.**
  - Each MIRI data model inherits the same behaviour and data structure as the underlying JWST data model but provides additional behaviour, such as:
    - Facilities for printing or plotting the contents of the data models (which makes the models more user-friendly when used interactively).
    - Facilities for masking the data arrays.
    - The ability to combine data models using mathematical operators.

Structure of a MIRI Data Model

Python class

miri_gain_model

```
49 class MiriGainModel(MiriDataModel, HasData):
50     """
51
52     A data model for MIRI gain data.
53
54     :Parameters:
55
56     init: shape tuple, file path, file object, pyfits.HDUList, numpy array
57         An optional initializer for the data model, which can have one
58         of the following forms:
59
60         * None: A default data model with no shape. (If a data array is
61           provided in the mask parameter, the shape is derived from the
62           array.)
63         * Shape tuple: Initialize with empty data of the given shape.
64         * File path: Initialize from the given file.
65         * Readable file object: Initialize from the given file obje
66         * pyfits.HDUList: Initialize from the given pyfits.HDULis
67
68     data: numpy array (optional)
69         An array containing the gain data.
70         If a data parameter is provided, its contents o rite the
71         data initialized by the init parameter.
72     \*\*kwargs:
73         All other keyword arguments are passed t e DataModel initialiser.
74         See the jwst.datamodels documentation  the meaning of these keywords.
75
76     """
77     schema_url = "miri_gain.schema.yaml"
78
79     def __init__(self, init=None, data=None, **kwargs):
80         """
81
82         Initialises the MiriGainModel class.
83
84         Parameters: See class doc string.
85
86         """
87         super(MiriGainModel, self).__init__(init=init, **kwargs)
88
89
90         # Data type is Gain.
91         self.meta.model_type = 'GAIN'
92         self.meta.reftype = 'GAIN'
93
94         # This is a reference data model.
95         self._reference_model()
96
97         # Update the data array if it has been specifically provided.
98         HasData.__init__(self, data)
```

YAML Schemas

miri_gain.schema.yaml

```
1 allOf:
2 - type: object
3   properties:
4     meta:
5       $ref: miri_metadata.schema.yaml
6 - $ref: http://jwst.stsci.edu/schemas/gain.schema.yaml
7
```

gain.schema.yaml

```
1 allOf:
2 - $ref: referencefile.schema.yaml
3 - $ref: subarray.schema.yaml
4 - $ref: keyword_gainfact.schema.yaml
5 - type: object
6   properties:
7     data:
8       title: The gain
9       fits_hdu: SCI
10      default: 0.0
11      ndim: 2
12      datatype: float32
13 $schema: http://stsci.edu/schemas/fits-schema/fits-schema
14
```

Each MIRI data model consists of a Python class associated with a YAML document describing the data structure. A MIRI YAML document usually inherits its structure from a JWST YAML document.

Structure of a MIRI Data Model

YAML Schemas

miri_metadata.schema.yaml

```
type: object
title: Top Level MIRI Metadata
properties:
  date:
    anyOf:
      - $ref
      - type
  title: D
  fits_key
  origin:
    title: O
    type: st
    fits_key
  flight_mod
    type: st
    title: V
    fits_key
  filename_o
    type: st
    title: O
    fits_key
  filetype:
    title: T
    type: st
    fits_key
  model_type
    title: T
    type: st
    fits_key
  version:
    type: st
    title: V
    fits_key
  observer:
    type: st
    title: P
    fits_key
  creator:
    type: st
    title: S
    fits_key
  telescope:
    type: st
    title: T
    default:
    enum: [J
    fits_key
  object:
    type: st
    title: N
```

miri_metadata.schema.yaml

```
instrument:
  type: object
  title: Information about the instrument and detectors
  properties:
    name:
      type: string
      title: Instrument used to acquire data
      default: MIRI
      enum: [MIRI]
      fits_keyword: INSTRUME
    model:
      type: string
      title: Instrument model name
      enum: [FM, VM, JPL, ANY, N/A]
      fits_keyword: MODELNAM
    filter:
      type: string
      title: Filter used by the instrument (imaging)
      enum: [F560W, F770W, F1000W, F1130W, F1280W, F1500W, F1800W, F2100W,
             F2550W, F2550WR, F1065C, F1140C, F1550C, F2300C, P750L, FLENS,
             FND, OPAQUE, ANY, N/A, '']
      fits_keyword: FILTER
    ccc_pos:
      type: string
      title: MIRI CCC position.
      enum: [OPEN, CLOSED, LOCKED]
      fits_keyword: CCC_POS
    calibmode:
      type: string
      title: Calibration source operating mode
      fits_keyword: CALMODE
    channel:
      type: string
      title: MIRI channel relevant (MRS)
      enum: ['1', '2', '3', '4', '12', '34', ANY, N/A, '']
      fits_keyword: CHANNEL
    dichroic_a:
      type: string
      title: MIRI dichroic wheel A setting (MRS)
      enum: [SHORT, MEDIUM, LONG, N/A, '']
      fits_keyword: DGAA
    dichroic_b:
      type: string
      title: MIRI dichroic wheel B setting (MRS)
      enum: [SHORT, MEDIUM, LONG, N/A, '']
      fits_keyword: DGAB
    band:
      type: string
      title: MIRI sub-channel relevant (MRS)
      enum: [SHORT, MEDIUM, LONG, SHORT-MEDIUM, SHORT-LONG, MEDIUM-SHORT,
```

miri_gain.schema.yaml

```
allOf:
- type: object
  properties:
    meta:
      $ref: miri_metadata.schema.yaml
- $ref: http://jwst.stsci.edu/schemas/gain.schema.yaml
```

gain.schema.yaml

```
allOf:
- $ref: referencefile.schema.yaml
- $ref: subarray.schema.yaml
- $ref: keyword_gainfact.schema.yaml
- type: object
  properties:
    data:
      title: The gain
      fits_hdu: SCI
      default: 0.0
      ndim: 2
      datatype: float32
$schema: http://stsci.edu/schemas/fits-schema/fits-schema
```

MIRI YAML schemas include a MIRI-specific metadata description, which means a MIRI data model can only contain metadata valid for MIRI.

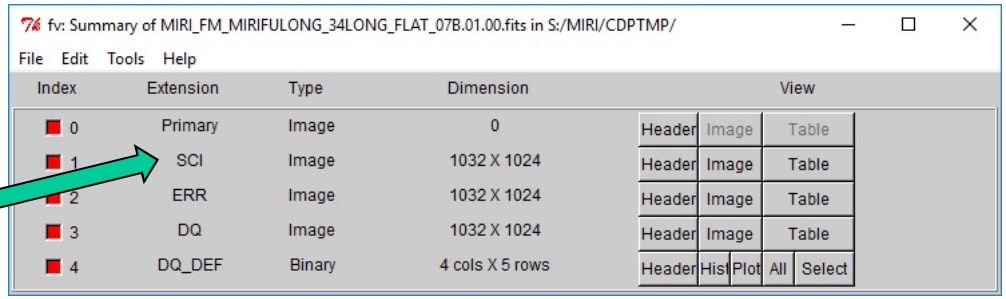
Relationship between YAML and FITS - 1

FLAT: data arrays

flat.schema.yaml

```
1 allOf:
2 - $ref: referencefile.schema.yaml
3 - $ref: subarray.schema.yaml
4 - $ref: keyword_filter.schema.yaml
5 - $ref: keyword_pfilter.schema.yaml
6 - $ref: keyword_pupil.schema.yaml
7 - $ref: keyword_ppupil.schema.yaml
8 - $ref: keyword_channel.schema.yaml
9 - type: object
10  properties:
11    data:
12      title: The science data
13      fits_hdu: SCI
14      default: 0.0
15      ndim: 2
16      datatype: float32
17    dq:
18      title: Data quality array
19      fits_hdu: DQ
20      default: 0
21      datatype: uint32
22    err:
23      title: Error array
24      fits_hdu: ERR
25      default: 0.0
26      datatype: float32
27    dq_def:
28      $ref: dq_def.schema.yaml
29 $schema: http://stsci.edu/schemas/fits-schema/fits-schema
30
```

FITS equivalent



Each data property defined within a YAML document is stored in a FITS extension.

Data array properties (such as data, err and dq) are stored in an image extension and table properties (such as dq_def) are stored in a binary table extension.

NOTE: Data models containing a SCI+ERR+DQ triplet (science data + error + data quality) are very common, and there is a MiriMeasuredModel data model designed specifically to handle such data models.

MIRI European Consortium

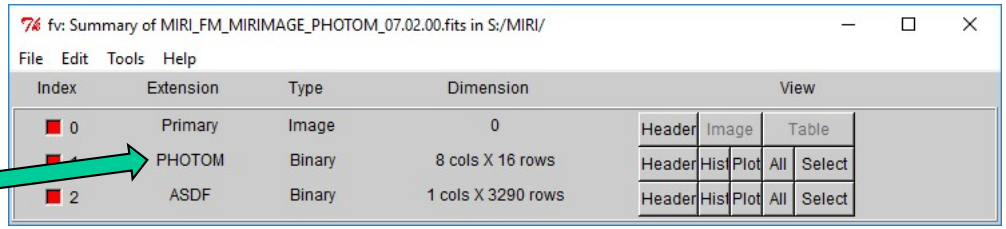
Relationship between YAML and FITS - 2

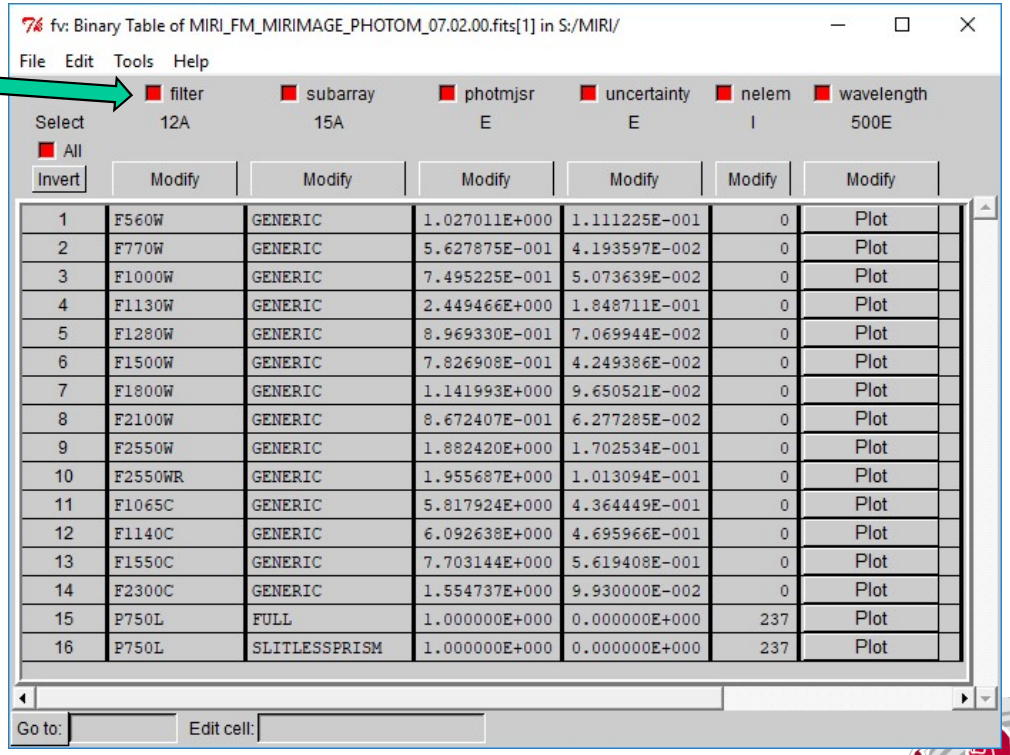
PHOTOM: data table

mirimg_photom.schema.yaml

```
1 title: MIRI imager photometric flux conversion data model
2 allOf:
3 - $ref: photom.schema.yaml
4 - $ref: keyword_pixelarea.schema.yaml
5 - type: object
6   properties:
7     phot_table:
8       title: Photometric flux conversion f          table
9       fits_hdu: PHOTOM
10      datatype:
11      - name: filter
12        datatype: [ascii, 12]
13      - name: subarray
14        datatype: [ascii, 15]
15      - name: photmjsr
16        datatype: float32
17      - name: uncertainty
18        datatype: float32
19      - name: nelem
20        datatype: int16
21      - name: wavelength
22        shape: [500]
23        datatype: float32
24      - name: relresponse
25        shape: [500]
26        datatype: float32
27      - name: relresperror
28        shape: [500]
29        datatype: float32
30 $schema: http://stsci.edu/schemas/fits-schema/fits-schema
31
```

FITS equivalent





MIRI European Consortium

Data Model Hierarchy

.meta
- .origin
- .version
- .telescope
- .instrument
  - .name, .model, .filter, .channel, .band, .detector, …
- .subarray
  - .name, .fastaxis, .slowaxis, .xstart, .ystart, .xsize, .ysize, …
- .exposure
  - .type, .readpatt, .nframes, …
- …

.data

.err

.dq

.dq_def

When a data model is stored as a Python object, its contents can be accessed as a hierarchy of attributes.

(see https://jwst-pipeline.readthedocs.io/en/latest/jwst/datamodels/index.html)

How to use a MIRI Data Model

- **Install MIRICLE, which includes the MiriTE package**
  - http://miri.ster.kuleuven.be/bin/view/Public/MirisimInstallation
  - http://www.miricle.org/
- **Import the required data model(s)**
  - from miri.datamodels.cdp import MiriGainModel
- **Either, open an existing file using the data model**
  - gain_model = MiriGainModel("MIRI_FM_MIRIMAGE_GAIN_04.00.00.fits")
- **Or, create a new data model from data**
  - data3x3 = np.array([[1.0,1.2,1.1],[1.3,1.2,1.0],[1.1,0.8,0.9]])
  - gain_model = MiriGainModel(data=data3x3)
- **Display the contents of the data model**
  - print(gain_model.info) # Show header info
  - print(gain_model.summary) # Show data summary
  - print(gain_model) # Show header and full contents
- **Change the contents, if necessary**
  - gain_model.meta.exposure.readpatt = 'N/A' # Correct 'ANY'->'N/A'
- **Save the data model to a new FITS file**
  - gain_model.save(("MIRI_FM_MIRIMAGE_GAIN_7B.00.00.fits")

How to use a MIRI Data Model – A Python session

```
(miricle.devel) smb@jura:/sw4/smb/MIRI$
(miricle.devel) smb@jura:/sw4/smb/MIRI$ python
Python 3.5.5 |Anaconda, Inc.| (default, May 13 2018, 21:12:35)
[GCC 7.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> from miri.datamodels.cdp import MiriGainModel
>>> gain_model = MiriGainModel("MIRI_FM_MIRIMAGE_GAIN_04.00.00.fits")
>>> print(gain_model.info)
MiriGainModel:
==============
Metadata
--------
Data model location                  FITS key   Value                  Comment
~~~~~~~~~~~~~~~~~~~                  ~~~~~~~~   ~~~~~                  ~~~~~~~
meta.author                        | AUTHOR   = JANE MORRISON        / Author of the reference file
meta.date                          | DATE     = 2015-07-02T16:48:19.198530 / Date this file was created (UTC)
meta.description                   | DESCRIP  = MIRI GAIN Array CDP4 / Description of the reference file
meta.exposure.nframes              | NFRAMES  = 1                    / Number of frames coadded in a group
meta.exposure.readpatt             | READPATT = ANY                  / Readout pattern
meta.filename                      | FILENAME = MIRI_FM_MIRIMAGE_GAIN_04.00.00.fits / Name of the file
meta.filename_original             | ORIGFILE = MIRI_FM_MIRIMAGE_GAIN_04.00.00.fits / Original name of the file
meta.instrument.detector           | DETECTOR = MIRIMAGE             / Name of detector used to acquire the data
meta.instrument.detector_settings  | DETSETNG = ANY                  / Detector settings used
meta.instrument.filter             | FILTER   = ANY                  / Filter used by the instrument (imaging)
meta.instrument.model              | MODELNAM = FM                   / Instrument model name
meta.instrument.name               | INSTRUME = MIRI                 / Instrument used to acquire the data
meta.model_type                    | DATAMODL = GAIN                 / Type of data model
meta.origin                        | ORIGIN   = MIRI European Consortium / Organization responsible for creating file
meta.pedigree                      | PEDIGREE = GROUND               / The pedigree of the reference file
meta.reftype                       | REFTYPE  = GAIN                 / Reference file type
meta.subarray.fastaxis             | FASTAXIS = 1                    / Fast readout axis direction
meta.subarray.name                 | SUBARRAY = GENERIC              / Subarray used
meta.subarray.slowaxis             | SLOWAXIS = 2                    / Slow readout axis direction
meta.subarray.xsize                | SUBSIZE1 = 1032                 / Number of pixels in axis 1 direction
meta.subarray.xstart               | SUBSTRT1 = 1                    / Starting pixel in axis 1 direction
meta.subarray.ysize                | SUBSIZE2 = 1024                 / Number of pixels in axis 2 direction
meta.subarray.ystart               | SUBSTRT2 = 1                    / Starting pixel in axis 2 direction
meta.telescope                     | TELESCOP = JWST                 / Telescope used to acquire the data
meta.useafter                      | USEAFTER =                      / Use after date of the reference file
meta.version                       | VERSION  = 04.00.00             / Version number of data found in file
HISTORY = 'see MIRI-TR-00005-UA-Gain_03.00.pdf for more details'
HISTORY = 'Created from: MiriGainModel'

>>>
```

General utilities

- **Generic data model opening function:**
  - from miri.datamodels import open
  - any_model = open("MIRI_FM_MIRIMAGE_GAIN_04.00.00.fits")
  - print(any_model.__class__.__name__) # Which model was used?
    MiriGainModel
  - print(any_model.info) # Show header info, etc…

- **Fetch a calibration data product:**
  - from miri.datamodels.cdplib import get_cdp
  - cdpmodel = get_cdp("GAIN", detector="MIRIMAGE")

    Please enter ftp password for miri:
    INFO:miri.cdplib.get_cdp:Reading 'GAIN' model from
    '/sw4/smb/MIRI/MIRI_FM_MIRIMAGE_GAIN_04.00.00.fits‘

  - print (cdpmodel.info) # Show header info, etc…

Useful CDP verification and checking commands

- **Check that a Calibration Data Product adheres to STScI standards:**
  - cdp_verify.py <filename>

    Useful as a final check before submitting a new CDP.

- **Display the contents of a Calibration Data Product:**
  - cdp_print.py <filename> [--info] [--summary]

    Useful for finding out the structure of a CDP data model.
    Use the --info and/or the --summary parameters to reduce the length of the output.

- **Fetch the documentation associated with a Calibration Data Product**
  - cdp_get_doc.py <filename>

    This works only if the “HISTORY DOCUMENT” parameter is defined correctly.

- **Find a more up to date CDP similar to this one.**
  - find_me_another.py <filename>

Data Model Functions – part 1

General functions for all models.

- **Data Display**
  - print(model)
    - Display contents of entire model.
  - print(model.info)
    - Display title and metadata
  - print(model.summary)
    - Display a summary of the data arrays and tables
  - model.plot()
    - Display a generic plot using matplotlib
- **Define Metadata**
  - model.set_observation_metadata()
  - model.set_pointing_metadata()
  - model.set_instrument_metadata()
  - model.set_exposure_metadata()
  - model.set_subarray_metadata()
  - model.set_wcs_metadata()
    - Set common collections of metadata keywords
  - model.add_history()
    - Add a history string to the metadata
  - model.set_referencefile_metadata()
  - model.add_referencefile_history()
    - Add reference file metadata in standard format.

Model-specific functions (examples).

- **MiriLinearityModel**
  - get_forward_table()
  - get_reverse_table()
    - Convert linearity coefficients into a forward or reverse translation table.
- **MiriMrsResolutionModel**
  - regenerate_phase1_spline()
  - regenerate_phase2_model()
  - regenerate_phase3_model()
    - Recreate the spline or polynomial functions described in the data model tables.
- **MiriRampModel**
  - plot_ramp()
    - Display the ramp(s) for specified rows and columns.
- **Etc…**

Using these functions reduces the likelihood of a mistake when creating a Calibration Data Product.

Data Model Functions – part 2

Functions specific to models containing a SCI+ERR+DQ triplet

- **Return masked or filled arrays**
  - model.set_data_fill('median')
    - Fill bad values in the SCI array (as indicated by the DQ array) with the min, max, mean or median value (default 'mean').
  - model.set_err_fill('max')
    - Fill bad values in the SCI array (as indicated by the DQ array) with the min, max, mean or median value (default 'max').
  - filled_data = model.data_filled
  - masked_data = model.data_masked
  - filled_err = model.err_filled
  - masked_err = model.err_masked
- **Mathematical operators**

  The following operations will combine the data arrays using the specified operator and combine the error arrays as vectors (sum of squares).
  - ratio_model = model1 / model2
  - summed_model = model1 + model2
  - multipled_model = model1 * model2
  - subtracted_model = model1 – model2

Data Model References

- **Description of JWST file names, formats and data structures (Jdox)**
  - https://jwst-docs.stsci.edu/display/JDAT/JWST+File+Names%2C+Formats%2C+and+Data+Structures
- **Description of JWST data models (readthedocs)**
  - https://jwst-pipeline.readthedocs.io/en/latest/jwst/datamodels/index.html
- **Description of MIRI data models**
  - http://miri.ster.kuleuven.be/bin/view/Internal/Software/MiriNewDataProductsImplementation
  - http://miri.ster.kuleuven.be/pub/Internal/Software/SoftDevDocs/miri_datamodels.pdf
- **Also see the README files contained within each GitHub folder:**
  - **https://github.com/JWST-MIRI/MiriTE/blob/master/README**
  - **https://github.com/JWST-MIRI/MiriTE/blob/master/datamodels/README**
  - **https://github.com/JWST-MIRI/MiriTE/blob/master/datamodels/scripts/README**

Ramps to Slopes CDPs and Data Models

Calibration Data Product	REFTYPE	MIRI Data Model	CRDS
Bad Pixel Mask	MASK	MiriBadPixelMaskModel	▶
Dark Correction	DARK	MiriDarkReferenceModel	▶
Linearity Correction	LINEARITY	MiriLinearityModel	▶
Pixel Saturation	SATURATION	MiriPixelSaturationModel	▶
Droop Correction	DROOP	MiriDroopModel	
Latent Correction	LATENT	MiriLatentDecayModel	
Jump Correction	JUMP	MiriJumpModel	
Gain	GAIN	MiriGainModel	▶
Read Noise	READNOISE	MiriReadnoiseModel	▶
Last Frame Correction	LASTFRAME	MiriLastFrameModel	▶
RSCD	RSCD	MiriResetSwitchChargeDecayModel	▶
Photometric Conversion Efficiency	PCE	MiriPceModel	

Imager CDPs and Data Models

Calibration Data Product	REFTYPE	MIRI Data Model	CRDS
Imager PSF and PSF-OOF	PSF or PSF-OOF	MiriImagingPointSpreadFunctionModel	
Imager Distortion	DISTORTION	MiriImagingDistortionModel	
Imager Pixel Area	AREA	MiriPixelAreaModel	▶
Imager Photometric Correction	PHOTOM	MiriImagingPhotometricModel	▶
Imager Colour Correction	COLCORR	MiriImagingColourCorrectionModel	
Imager Colour Correction (Power Law)	COLCORRPL	MiriPowerlawColourCorrectionModel	
Imager Pixel Flat or Combined Flat	PIXELFLAT or FLAT	MiriFlatfieldModel	▶
Imager Sky Flat	SKYFLAT	MiriSkyFlatfieldModel	▶

LRS CDPs and Data Models

Calibration Data Product	REFTYPE	MIRI Data Model	CRDS
LRS PSF and PSF Monochromatic	PSF or PSF-MONOCHROM	MiriLrsPointSpreadFunctionModel	
LRS Distortion and Wavelength	SPECWCS	MiriLrsD2WModel	
LRS Pixel Flat	PIXELFLAT or FLAT	MiriFlatfieldModel	▶
(LRS SRF)	SRF	MiriLrsFluxconversionModel	

MRS CDPs and Data Models

Calibration Data Product	REFTYPE	MIRI Data Model	CRDS
MRS PSF	PSF	MiriMrsPointSpreadFunctionModel	
MRS Distortion and Wavelength	DISTORTION	MiriMrsDistortionModel12 and MiriMrsDistortionModel34	
MRS Photometric Correction	PHOTOM	MiriMrsFluxconversionModel	▶
MRS Pixel Flat	PIXELFLAT or FLAT	MiriFlatfieldModel	▶
MRS Fringe Flat	FRINGE	MiriFringeFlatfieldModel	▶
MRS Straylight	STRAY	MiriMrsStraylightModel	▶
MRS Wavelength Offsets	WAVCORR	MiriMrsWavelengthCorrectionModel	
MRS Transmission Correction	TRACORR	MiriMrsTransmissionCorrectionModel	
MRS Fringe Frequencies	FRINGEFREQ	MiriMrsFringeFrequenciesModel	
MRS Aperture Correction	APERCORR	MiriMrsApertureCorrectionModel	
MRS Spectral Resolution	RESOL	MiriMrsResolutionModel	▶

MIRI and JWST Data Model "Features"

- **If a FITS header keyword is not recognised within a data model, it is put into an "extra FITS" storage area. The keyword is not accessible from the data model, but it will be preserved when the data model is saved to a new file.**
  - If an expected header keyword is not present in the ".info" list, or if you attempt to change a keyword but it stubbornly remains the same, please let me know and I will add it to the data model definition.
- **The contents of every file are checked against the expected structure of the data model.**
  - You may see warning messages, or the file may not open at all.
  - This gives an early warning if something is wrong inside the file.
- **The JWST data model software has control over the file format.**
  - This ensures our files are compatible with the JWST software.
  - But the JWST software might create extra keywords or HDUs (e.g. "METADATA" and "ASDF") that we don't want.
    - The most recent version of the JWST software makes the creation of "ASDF" optional.